Tracé de courbes avec *Code_Aster*

Résumé :

Ce document explique comment, à partir d'un résultat de calcul, on peut produire des tables ou des fonctions, extraire les valeurs de ces tables ou fonctions, les manipuler, et enfin en dernier lieu utiliser les commandes d'impression pour tracer des courbes.

Si vous disposez d'une fonction ou d'une table et que vous souhaitez simplement la représenter sous forme d'une courbe, allez directement au paragraphe [§4], et consultez les documentations des commandes `IMPR_FONCTION` [U4.33.01] et `IMPR_TABLE` [U4.91.03].

Remarque préliminaire

Le post-traitement doit être effectué en POURSUITE et non pas à la suite du calcul. Plusieurs raisons à cela :

1) *en cas d'erreur, on ne perd pas les heures de calcul qui ont permis d'atteindre le résultat,*
2) *on peut effectuer de nombreux post-traitements directement en lançant Stanley sur la base résultat du calcul (voir astk [U1.04.00] ou STANLEY [U4.81.31]),*
3) *pour simplifier le post-traitement, on peut utiliser les possibilités offertes par Python qui nécessitent d'être en PAR_LOT='NON' dans POURSUITE ce qui empêche l'utilisation d'eficas pour ce type de post-traitement, alors qu'il est plus simple d'utiliser eficas pour construire le jeu de données principal.*

1 Extraire les données à l'aide des commandes Aster

On suppose que l'utilisateur dispose d'un résultat de calcul obtenu à partir, par exemple, de la commande MODE_ITER_SIMULT pour un calcul de modes propres, STAT_NON_LINE pour un calcul statique non linéaire, ou DYNA_NON_LINE pour un calcul dynamique non linéaire…

On dispose dans tous les cas d'un concept résultat qui sera par exemple de type mode_meca, mode_flamb, dyna_trans, tran_gene, evol_elas, evol_noli, evol_ther, etc. selon la commande utilisée et qui contient des champs de valeurs que l'on souhaite représenter sous forme de courbes.

1.1 Produire une fonction ou une table

Rappel

Une fonction est composée de deux listes de valeurs, abscisses et ordonnées ; les abscisses sont nécessairement monotones.
Une table est un agglomérat de valeurs non nécessairement de même type auxquelles on accède via un paramètre, « nom de colonne ». Dans l'utilisation des tables qui nous intéresse ici, on produira généralement des colonnes de nombres réels ; leur variation est quelconque.
Pour plus de détails sur ce qu'est une table, on pourra consulter la documentation de IMPR_TABLE [U4.91.03].

Les valeurs peuvent être extraites en utilisant les commandes suivantes :

- RECU_FONCTION [U4.32.03] : produit une fonction à partir d'un résultat, d'un champ, d'une table… Exemple : évolution temporelle d'une composante d'un champ en un point particulier.
- POST_RELEVE_T [U4.81.21] : produit une table à partir d'un résultat ou d'un champ. On peut extraire une quantité associée aux composantes d'un champ (une composante, un invariant…) en certains points particuliers ou le long d'un chemin non nécessairement rectiligne.
- MACR_LIGN_COUPE [U4.81.13] : produit une table à partir d'un résultat ou d'un champ le long d'une ligne de coupe (ligne droite composée de segments réguliers).
- RECU_TABLE [U4.71.02] : produit une table à partir des valeurs d'un ou plusieurs paramètres d'un résultat. Par exemple : l'évolution du paramètre de pilotage au cours d'un calcul. On peut aussi extraire une table de quelques structures de données particulières.
- CREA_CHAMP [U4.72.04] : permet d'extraire un champ d'une structure de données résultat. Ceci peut être utile quand une commande ne sait pas traiter certains résultats. On peut par exemple ensuite récupérer une fonction via RECU_FONCTION/CHAM_GD.

1.2 Traiter des blocs de données

Un bloc de données est simplement un tableau de valeurs à N lignes, P colonnes.

Dans certains cas, on dispose d'un ou plusieurs fichiers composés chacun d'un ou plusieurs blocs de données, séparés par des lignes de texte. Pour construire des fonctions à partir de tels fichiers, on peut utiliser `LIRE_FONCTION` [U4.32.02].

Par exemple, on effectue une étude paramétrique, chaque calcul produit un tableau qui vient enrichir un fichier de résultat ; le fichier peut aussi avoir été construit par un calcul unique ou encore manuellement, peu importe. Un tel fichier pourrait ressembler à cela :

```
AVEC PARA=1.23
INST          COOR_X        DY
 1.00000E+00  1.00000E+01 -3.02717E-2
 1.20000E+00  1.00000E+01 -8.14498E-2
 1.40000E+00  1.00000E+01 -7.97278E-2
 1.60000E+00  1.00000E+01 -3.86827E-2
 1.80000E+00  1.00000E+01 -8.48309E-2
 2.00000E+00  1.00000E+01 -9.37561E-2
 2.20000E+00  1.00000E+01  7.18293E-2
 2.40000E+00  1.00000E+01  6.05322E-2

AVEC PARA=1.98
INST          COOR_X        COOR_Y        DX            DY
 1.00000E+00  1.00000E+01  0.00000E+00 -3.02717E-2  2.07127E-01
 1.10000E+00  1.00000E+01  0.00000E+00 -8.14498E-2  4.14928E-01
 1.20000E+00  1.00000E+01  0.00000E+00 -7.97278E-2  7.92728E-01
 1.80000E+00  1.00000E+01  0.00000E+00 -7.86827E-2  6.88227E-01
 2.45000E+00  1.00000E+01  0.00000E+00  8.48309E-2  3.43029E-01
```

On a plusieurs blocs qui n'ont pas forcément le même nombre de colonnes.

Supposons que l'on veuille comparer le déplacement `DY` obtenus avec les deux valeurs de `PARA`, on utilisera par exemple :

```
fDY1=LIRE_FONCTION(
   TYPE='FONCTION',
   INDIC_PARA=(1,1,),        # les abscisses sont prises dans le bloc 1, colonne 1
   INDIC_RESU=(1,3,),        # les ordonnées sont prises dans le bloc 1, colonne 3
   UNITE=38,
   NOM_PARA='INST',
   NOM_RESU='DY',)

fDY2=LIRE_FONCTION(
   TYPE='FONCTION',
   INDIC_PARA=(2,1,),        # les abscisses sont prises dans le bloc 2, colonne 1
   INDIC_RESU=(2,5,),        # les abscisses sont prises dans le bloc 2, colonne 5
   UNITE=38,
   NOM_PARA='INST',
   NOM_RESU='DY',)
```

tracé classique de deux fonctions avec `IMPR_FONCTION` :

```
IMPR_FONCTION(FORMAT='XMGRACE',
              UNITE=29,
              COURBE=(_F(FONCTION=fDY1,
                         LEGENDE='PARA=1.23',),
                      _F(FONCTION=fDY2,
                         LEGENDE='PARA=1.98',),),
              TITRE='DY=f(t)',)
```

2 Transformer les valeurs d'une table ou d'une fonction en objets Python

Remarque

> *Les traitements en Python qui sont utilisés à partir de maintenant imposent d'être en `PAR_LOT='NON'` dans `POURSUITE` (ou `DEBUT`).*

L'objet est ici de récupérer les valeurs d'une table ou d'une fonction dans un objet Python pour les manipuler ensuite. Notons qu'il est parfois pratique de produire une fonction à partir des données d'une table, en filtrant éventuellement certaines lignes de la table ; c'est une utilisation de `RECU_FONCTION` [U4.32.03], que nous n'aborderons pas ici.

Sur les objets de type fonction, on dispose des méthodes :

1) `Valeurs` pour récupérer les abscisses et les ordonnées dans 2 listes Python de réels.

Avec les données du paragraphe précédent :

```
>>> lx, ly = fDY2.Valeurs()
```

On obtient :

```
>>> print lx
[1.0, 1.1, 1.2, 1.8, 2.45]
>>> print ly
[0.207127, 0.414928, 0.792728, 0.688227, 0.343029]
```

1) `Absc` et `Ordo` permettent de récupérer les abscisses et les ordonnées séparément.

```
>>> lx = fDY2.Absc()
>>> lx = fDY2.Ordo()
```

On peut accéder au contenu d'une cellule d'une table avec [*nom_paramètre*, *numéro_ligne*] :

```
>>> print tab['DY', 2]
-8.14498E-2
```

On peut également transformer l'objet (`JEVEUX`) table en une instance de la classe Python Table.

```
>>> tabpy = tab.EXTR_TABLE()
```

Le document [U1.03.02] détaille les méthodes Python disponibles sur les objets de type Table. Pour l'extraction des valeurs, on dispose notamment d'un méthode `values()` qui retourne un dictionnaire dont les clés sont les noms de paramètres (ex `'DY'`) et les valeurs les listes des valeurs de la table.

Attention

> Les listes Python sont indexées de 0 à $n-1$ (pour n éléments), l'équivalent de `tab['DY', 2]` est donc `tabpy['DY'][1]` !

Pour le tracé de courbes à partir de listes de réels Python, voir [§4].

3 Manipuler les valeurs en Python

On donne ici quelques exemples de manipulation des valeurs obtenues précédemment sous forme de listes ou de tableaux Numeric.

Numeric est un module Python optionnel (c'est à dire non inclus à la distribution de Python fournie sur www.python.org) mais indispensable pour utiliser *Code_Aster*, on peut donc faire `import Numeric` sur toutes installations de *Code_Aster*.

3.1 Avec des listes Python

Les listes Python sont facilement manipulables en utilisant les boucles. Prenons l'exemple de la notice [U2.51.01] pour $-10. < x < 10.$ en 100 pas :

$$\begin{cases} y=-1.5 & \text{si } \sin(x)<0 \\ y=-5. & \text{sinon} \end{cases}$$

On cherche donc à construire deux listes de réels, lx et ly .

Comme toujours en Python, il y a plusieurs moyens de faire, plus ou moins simplement, plus ou moins élégamment !

```
x0=-10.
pas=20./100
lx=[]
for i in range(101):
   lx.append(x0+i*pas)
```

ou bien

```
lx=[x0+i*pas for i in range(101)]
```

```
def f(x):
   if sin(x)<0.:
      return -1.5
   else:
      return -5.

ly=map(f, lx)
```

qui applique la fonction f à tous les éléments de lx

On peut tracer cette courbe en utilisant les mots-clés `ABSCISSE` et `ORDONNEE` de `IMPR_FONCTION` (cf. [§4]).

3.2 Avec des tableaux Numeric

La manipulation des données sous forme de tableaux Numeric est simplifiée par l'utilisation d'opérations très performantes (appelées *ufunc*) sur le tableau entier (dans l'exemple suivant on utilise `sin()`). Numeric gère également les tableaux à plusieurs dimensions.

En reprenant l'exemple précédent :

```
from Numeric import *
lx = arrayrange(-10., 10.+0.2, 0.2, Float)
ly = array(map(f, lx))
```

ou bien sans utiliser `f` :

```
ly = -1.5*less(sin(lx), 0.) + (-5.)*(1.-less(sin(lx), 0.))
```

Notons que `map()` retourne une liste et non un tableau Numeric. La deuxième expression est entre 10 et 20 fois plus rapide sur de *très* gros tableaux ($10^5 - 10^6$ termes), ce qui est cependant assez peu souvent le cas des fonctions ou tables issues d'*Aster*.

On peut tracer cette courbe en utilisant les mots-clés `ABSCISSE` et `ORDONNEE` de `IMPR_FONCTION` en prenant soin de convertir les tableaux Numeric en liste, par exemple (cf. [§4]) :

```
ABSCISSE = lx.tolist()
```

4 Exemples d'utilisation de IMPR_FONCTION/IMPR_TABLE

4.1 Fonction reconstruite à partir de deux listes Python

Exemple où l'on recale les résultats obtenus pour pouvoir les comparer à une solution de référence (les abscisses sont inversées, il faut comparer la valeur absolue, [SSLS501a]), on utilise les méthodes permettant d'extraire les abscisses et les ordonnées d'une fonction :

```
IMPR_FONCTION(FORMAT='XMGRACE',
              UNITE=53,
              COURBE=_F(ABSCISSE=[57766.1-x for x in MTAST.Absc()],
                        ORDONNEE=[abs(y)    for y in MTAST.Ordo()],),
              TITRE='Courbe recalée',
              LEGENDE_X='P2',
              LEGENDE_Y='MT',)
```

4.2 Tracé d'un résultat en fonction d'un autre

Cet exemple est extrait en partie du cas-test [FORMA03a], il s'agit d'une plaque trouée en traction. Après un calcul réalisé avec STAT_NON_LINE, on souhaite tracer l'effort résultant de traction en fonction du déplacement vertical moyen de la partie supérieure de l'éprouvette.

Détails du post-traitement :

```
POURSUITE()

SOLNL2=CALC_CHAMP(                         Calcul des forces nodales
   reuse    = SOLNL2, RESULTAT = SOLNL2,
   OPTION   = 'FORC_NODA',)

M=DEFI_GROUP(                              Définition du groupe de nœuds de
   reuse=M,                                dépouillement, la ligne supérieure de la plaque
   MAILLAGE=M,
   CREA_GROUP_NO=_F(GROUP_MA = 'LFG',
                    NOM      = 'LIGNE',),
)

tab=POST_RELEVE_T(
   ACTION=(
      _F(INTITULE   = 'Umoyen',            Relevé du déplacement moyen à tous les
         RESULTAT   = SOLNL2,              instants du calcul
         NOM_CHAM   = 'DEPL',
         NOM_CMP    = 'DY',
         TOUT_ORDRE = 'OUI',
         GROUP_NO   = 'LIGNE',
         OPERATION  = 'MOYENNE',
      ),
      _F(INTITULE   = 'Fresultante',       Relevé de l'effort résultant
         RESULTAT   = SOLNL2,
         NOM_CHAM   = 'FORC_NODA',
         TOUT_ORDRE = 'OUI',
         GROUP_NO   = 'LIGNE',
         RESULTANTE = 'DY',
         OPERATION  = 'EXTRACTION',),),)
```

```
IMPR_TABLE(TABLE=tab,)
```

Juste pour repérer le nom des paramètres

Contenu (partiel, limité aux 3 premiers instants) de la table :

```
INTITULE         NOEUD    RESU     NOM_CHAM           NUME_ORDRE INST         DY            SIYY          QUANTITE
 Umoyen           -        SOLNL2   DEPL                       1  1.00000E+00  2.38745E-02 -              MOMENT_0
 Umoyen           -        SOLNL2   DEPL                       1  1.00000E+00 -3.70291E-04 -              MOMENT_1
 Umoyen           -        SOLNL2   DEPL                       1  1.00000E+00  2.36709E-02 -              MINIMUM
 Umoyen           -        SOLNL2   DEPL                       1  1.00000E+00  2.40291E-02 -              MAXIMUM
 Umoyen           -        SOLNL2   DEPL                       1  1.00000E+00  2.40597E-02 -              MOYE_INT
 Umoyen           -        SOLNL2   DEPL                       1  1.00000E+00  2.36894E-02 -              MOYE_EXT
 Umoyen           -        SOLNL2   DEPL                       2  1.20000E+00  2.86494E-02 -              MOMENT_0
 Umoyen           -        SOLNL2   DEPL                       2  1.20000E+00 -4.44349E-04 -              MOMENT_1
 Umoyen           -        SOLNL2   DEPL                       2  1.20000E+00  2.84050E-02 -              MINIMUM
 Umoyen           -        SOLNL2   DEPL                       2  1.20000E+00  2.88350E-02 -              MAXIMUM
 Umoyen           -        SOLNL2   DEPL                       2  1.20000E+00  2.88716E-02 -              MOYE_INT
 Umoyen           -        SOLNL2   DEPL                       2  1.20000E+00  2.84273E-02 -              MOYE_EXT
 Umoyen           -        SOLNL2   DEPL                       3  1.40000E+00  3.34244E-02 -              MOMENT_0
 Umoyen           -        SOLNL2   DEPL                       3  1.40000E+00 -5.18504E-04 -              MOMENT_1
 Umoyen           -        SOLNL2   DEPL                       3  1.40000E+00  3.31393E-02 -              MINIMUM
 Umoyen           -        SOLNL2   DEPL                       3  1.40000E+00  3.36410E-02 -              MAXIMUM
 Umoyen           -        SOLNL2   DEPL                       3  1.40000E+00  3.36837E-02 -              MOYE_INT
 Umoyen           -        SOLNL2   DEPL                       3  1.40000E+00  3.31652E-02 -              MOYE_EXT
   [...]
 Fresultante      -        SOLNL2   FORC_NODA                  1  1.00000E+00  2.50000E+03 -              -
 Fresultante      -        SOLNL2   FORC_NODA                  2  1.20000E+00  3.00000E+03 -              -
 Fresultante      -        SOLNL2   FORC_NODA                  3  1.40000E+00  3.50000E+03 -              -
   [...]
```

```
Dy=RECU_FONCTION(
   TABLE = tab,
   PARA_X = 'INST',
   PARA_Y = 'DY',
   FILTRE = (
      _F(NOM_PARA = 'INTITULE',
         VALE_K   = 'Umoyen',),
      _F(NOM_PARA = 'QUANTITE',
         VALE_K   = 'MOMENT_0',),),)
```

Filtrage de la table pour en extraire $Dy=f(t)$

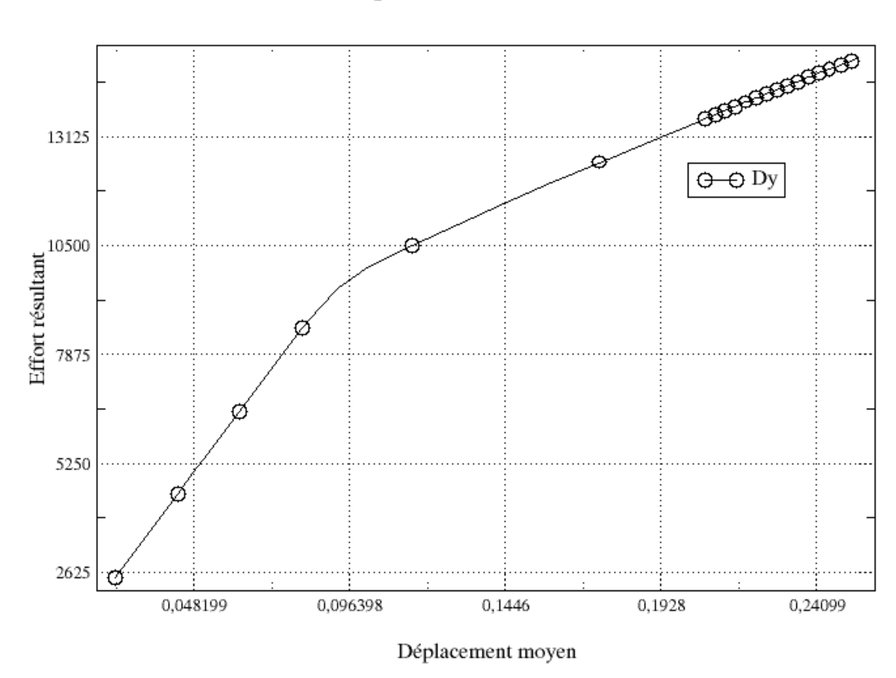
```
Fy=RECU_FONCTION(
   TABLE = tab,
   PARA_X = 'INST',
   PARA_Y = 'DY',
   FILTRE = (
      _F(NOM_PARA = 'INTITULE',
         VALE_K   = 'Fresultante',),),)
```

Filtrage de la table pour en extraire $Fy=f(t)$

```
IMPR_FONCTION(
   UNITE  = 29,
   FORMAT = 'XMGRACE',
   COURBE = (
      _F(FONC_X = Dy,
         FONC_Y = Fy,),),
   TITRE     = 'Plaque trouée en traction',
   LEGENDE_X = 'Déplacement moyen',
   LEGENDE_Y = 'Effort résultant',)
```

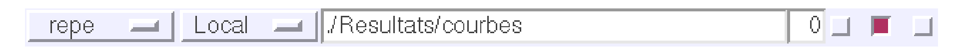
Tracé des ordonnées de Fy en fonction de Dy

Ce qui nous donne la courbe suivante :



4.3 Tracé d'un grand nombre de courbes

Dans certaines applications, on est amené à tracer de nombreuses courbes. Supposons que l'on souhaite comparer nos résultats à 50 points de mesure obtenus par ailleurs. Il serait alors fastidieux de définir 50 fichiers dans astk et d'utiliser 50 unités logiques différentes dans le fichier de commandes !
Il suffit alors d'utiliser le type « `repe` » en résultat dans astk (cf. [U1.04.00]) :



On procède ensuite ainsi dans le fichier de commandes en `PAR_LOT='NON'` dans `POURSUITE` :

```
# définition des 50 groupes de nœuds de dépouillement
lgrno = [ 'point01', 'point02', ..., 'point50' ]

# pour chaque nœud de dépouillement...
for point in lgrno :
   unit = 29
   # les fichiers DEPL_point0i.dat seront recopiés dans Resultats/courbes/
   DEFI_FICHIER(UNITE=unit, FICHIER='./REPE_OUT/DEPL_'+point+'.dat')

   tab=POST_RELEVE_T(
      ACTION=_F(INTITULE   = 'VonMises',
                RESULTAT   = resM,
                NOM_CHAM   = 'SIEQ_ELNO',
                NOM_CMP    = 'VMIS',
                TOUT_ORDRE = 'OUI',
                GROUP_NO   = point,
                OPERATION  = 'EXTRACTION',),)

   IMPR_TABLE(
```

```
UNITE  = unit,
TABLE  = tab,
FORMAT = 'XMGRACE',
NOM_PARA = ('INST', 'VMIS'),)
```

on “libère” l’unité pour la réutiliser pour la courbe suivante
```
DEFI_FICHIER(UNITE=unit, ACTION='LIBERER')
```

5 Quelques astuces utiles

On propose ici quelques manipulations des données des tables en Python récurrentes lorsque de l’on veut aller plus loin dans la génération de courbes depuis *Code_Aster*.

Extraire de la table issue de POST_RELEVE_T la liste des nœuds de post-traitement :

Lorsque l’on post-traite une grandeur sur un groupe de nœuds (à plus d’un nœud) pour plusieurs instants, les nœuds apparaissent pour chaque instant, il est donc nécessaire d’extraire la liste de ces nœuds sans répétition.

`tabpy = tab.EXTR_TABLE()`	Création de l’objet Table Python
`tno   = tabpy.NOEUD.values()`	On extrait les valeurs de la colonne `NOEUD`
`lno   = list(set([i.strip() for i in tno]))`	Méthode performante pour éliminer les doublons utilisant set
`lno.sort()`	Tri par ordre croissant

Construire un ou plusieurs mots-clés dynamiquement :

Ceci peut notamment être utile pour renseigner le mot-clé facteur `FILTRE` d’`IMPR_TABLE` en fonction du contexte ; on construit dans ce cas un dictionnaire qui est ensuite fourni en argument de la commande.

Ceci :
```
IMPR_TABLE(
   UNITE  = unit,
   TABLE  = tab,
   FORMAT = 'XMGRACE',
   FILTRE = (_F(NOM_PARA='NOEUD',
                VALE_K = 'N4',),
             _F(NOM_PARA='NOEUD',
                VALE_K = 'N4',),),
   NOM_PARA = ('INST', 'VMIS'),
)
```
est équivalent à cela :
```
mfac = {'FILTRE'   : [{'NOM_PARA':'NOEUD',     'VALE_K'  : 'N4'},
                      {'NOM_PARA':'INTITULE', 'VALE_K'  : 'exemple'},],
        'NOM_PARA' : ['INST', 'VMIS'],
        'UNITE'    : unit,
        'FORMAT'   : 'XMGRACE',}

IMPR_TABLE(
   TABLE  = tab,
   **mfac
)
```

L’intérêt étant bien entendu de pouvoir construire le dictionnaire comme on le souhaite.

Remarque

Les mots-clés facteurs (ici `FILTRE`) peuvent être construits en utilisant `_F(mot_clé = valeur)` , mais il est plus souple de les voir comme une liste de dictionnaires.